CHAPTER 5

Recent Developments for Modeling

In addition to the architectures and approaches identified by Pew and Mavor (1998), there are a few other architectures that are worth examining. In this chapter we note them, including the lessons they provide. Our reviews also explicitly consider ease of use (i.e., model populating).

We focus our comments on cognitive architectures because they have been created for modeling the strengths and limitations of human behavior. Any system built for other reasons that was adapted in this way—for example, other AI systems—would start to approach these systems in capabilities and limitations. It is quite likely that the cognitive architecture that best matches human behavior will vary by the type of behavior and level of aggregation. For example, different architectures will be preferred for modeling a soldier performing simple physical tasks than for a deliberate and reflective commander.

There will continue to be a range of architectures created. We agree completely with Pew and Mavor (pp. 110-111) that further work is necessary before settling on an architecture. That is not to say that architectures will not continue to converge (e.g., Soar and EPIC, Chong, 2001, and Soar and ACT-R, Jones, 1998). We start, however, by examining ways to summarize data and some advanced AI techniques to help create models. We then examine several architectures.

5.1 Data Gathering and Analysis Techniques

Scattered throughout Pew and Mavor (e.g., pp. 323-325) are comments about the need for data to develop and test models. Data to develop models can come from a wide variety of sources. Data can come from speaking to experts and having them do tasks off-line, so-called knowledge acquisition (Chipman & Meyrowitz, 1993; Schraagen, Chipman, & Shalin, 2000; Shadbolt & Burton, 1995). Data can also come from having experts talk aloud while performing the task (Ericsson & Simon, 1993). Talking aloud is a more accurate way to acquire the knowledge because it is based on actual behavior rather then someone's impression and memory of behavior. It is, however, a more costly approach because the modeler must infer the behavior generators. Data for developing models can also come from non-verbal measurements of experts while they perform the task. Non-verbal measurements are probably the least useful data (but still useful in some circumstances) for developing models. These data are useful, however, in testing models that make timing predictions. Data can also come from previously run studies, reviews, and compendia of such studies (e.g., Boff & Lincoln, 1988 SeKular & Blake, 1994). A useful review of data types and analysis methods in this area is provided by Hoffman (1987).

A major requirement will be a balance between the experimental control of the lab and the richness of the real world. An appropriate balance can sometimes be achieved by gathering data in the same micro-world simulations in which the models will be deployed, such as synthetic environments. These environments can be used to model all the salient aspects of the real world, while still providing some level of experimental control.

Once the data are in hand, they will often have to be aggregated or summarized. Expert summaries from knowledge acquisition already represent summarized data, but the field of verbal protocol analysis has developed a wide range of techniques for summarizing such data.

Reviews and suggestions in this area are available (Ericsson & Simon, 1993; Sanderson & Fisher, 1994), but there exists a very wide range of techniques that vary based on how advanced the theory is, the purposes of the research, and the domain. Survival analysis is one example of an advanced technique to examine protocol data for temporal patterns for later inclusion and comparison against model behavior (Kuk, Arnold, & Ritter, 1999).

With data in hand, the next step is either to develop a model or to test an existing model. There is little formal methodology about how to create models. Some textbooks attempt to teach this creative task either directly (vanSomeren, Barnard, & Sandberg, 1994) or by example (McClelland & Rumelhart, 1988; Newell & Simon, 1972). There are summaries of the testing process (Ritter & Larkin, 1994) and of some possible tests (Ritter, 1993a). Tenney and Spector (2001); and Ritter and Bibby (2001) provide particularly useful example sets of comparisons. Repairing a model based on the results of the tests can be a task requiring a lot of creativity.

5.2 Advanced AI Approaches

There are some existing AI tools that could be used to create, augment, or optimize models of performance. We note here three tools with which we are particularly familiar. These include approaches for creating behaviors, such as genetic algorithms and traditional AI-planning programs.

5.2.1 Genetic Algorithms

Genetic Algorithms (GAs) are search methods that can be used in domains in which no heuristic knowledge is available and an objective function exhibits high levels of incoherence (Goldberg, 1989). That is to say, a small change to the solution state may often result in large changes to the objective function or fitness measure. These algorithms are expensive in machine resources and exhibit slow (but often steady) convergence to a solution. They might be used as a search strategy of last resort for plan formation.

GAs are a family of algorithms loosely based on Darwinian evolution. They optimize functions without assuming that the search space will be linear. They start with a population of templates for possible solutions (analogous to sets of chromosomes), and evaluate them to determine how well they perform (fitness). After the fitness values are computed, a new population is created. A variety of methods have been used to create the next generation, but in each case the underlying principle has been to include copies of the chromosomes proportional to their fitness, and at each generation to create new combinations by combining two parents' chromosomes. The cycles of evaluation and creation are then repeated.

Heuristics can be used with GAs to seed the initial population in a non-random way or to guide the crossover process in a way that changes the distribution of offspring. Using heuristics results in a *memetic* algorithm (one that manipulates basic blocks of information or *memes*). As has been common experience throughout the history of AI, this introduction of domain knowledge can drastically transform the performance of the GA. Such algorithms have been found to exceed the performance of previous approaches in a number of domains (Burke, Elliman, & Weare, 1995). There may be reason for using GAs as a search strategy in planning.

5.2.2 Tabu Search

Tabu search, as developed by Glover (Glover & Laguna, 1998), is a general purpose approach remarkably effective for difficult problems where the objective function has some local coherence. It is surprising how often hill-climbing approaches such as the A* algorithm are used in current plan-building algorithms, despite the domains being prone to local maxima. Tabu search uses the novel concept of *recency* memory to prevent moves in a solution space from being tried when some component of that state has recently been changed in a previous move. This surprisingly simple idea forces the search away from a local maximum. Long-term memory is used to hold the best solution state found so far and this knowledge may be used to restart the search far away from any previous exploration of the state space.

The Tabu search approach would almost certainly lead to improved solutions with reasonable computational complexity. It would be worth using this approach to search for strategies and plans at various levels in a synthetic environment from the individual combatant to the highest level source of command and control.

Soar is impressive in its ability to reuse parts of problems that have been solved in the past and to plan in a goal-directed way that can seem ingenious. Real human problem solving can be less structured, however, and can leap from one approach to another in a manner that is difficult to model. Tabu search has this characteristic, however, as part of its diversification strategy. Including Tabu search in a cognitive architecture would be interesting. There may be some advantages to be gained by grafting on other similar systems that modify the beliefs of a cognitive architecture so as to maintain various types of logical consistency in the set of facts held.

5.2.3 Multiple Criteria Heuristic Search[1]

Heuristic search, one of the classic techniques in AI, has been applied to a wide range of problem-solving tasks including puzzles, two-player games, and path-finding problems. A key assumption of all problem-solving approaches based on utility theory, including heuristic search, is that we can assign a single utility or cost to each state. This, in turn, requires that all criteria of interest can be reduced to a common ratio scale.

[1]This section was drafted by Brian Logan and revised by the authors.

The route-planning problem has conventionally been formulated as one of finding a minimum-cost (or low-cost) route between two locations in a digitized map, where the cost of a route is an indication of its quality (e.g., Campbell, Hull, Root, & Jackson, 1995). In this approach, planning is regarded as a search problem in a space of partial plans, allowing many of the classic search algorithms such as A* (Hart, Nilsson, & Raphael, 1968) or variants such as A*epsilon (Pearl, 1982) to be applied. However, while such planners are complete and optimal (or optimal to some bound e), formulating the route-planning task in terms of minimizing a single criterion is difficult.

For example, consider the problem of planning a route in a complex terrain of hills, valleys, impassable areas, and so on. A number of factors will be important in evaluating the quality of a plan: the length of the route, the maximum negotiable gradient, the degree of visibility, and so on. In any particular problem, some of these criteria will affect the feasibility of the route, while others are simply preferences. Route planning is an example of a wide class of multi-criteria, problem-solving tasks, where different criteria must be traded off to obtain an acceptable solution.

One way of incorporating multiple criteria into the problem-solving process is to define a cost function for each criterion and use, for example, a weighted sum of these functions as the function to be minimized. We can, for example, define a *visibility cost* for being exposed and combine this cost with cost functions for the time and energy required to execute the plan to form a composite function that can be used to evaluate alternative plans. However, the relationship between the weights and the solutions produced is complex in reality, and it is often unclear how the different cost functions should be combined linearly as a weighted sum to give the desired behavior across all magnitude ranges for the costs. This makes it hard to specify what kinds of solutions a problem-solver should produce and hard to predict what a problem solver will do in any given situation; small changes in the weight of one criterion can result in large changes in the resulting solutions. Changing the cost function on a single criterion to improve the behavior related to that criterion often leads to changing all the weights for all the other costs as well because the costs are not independent. Moreover, if different criteria are more or less important in different situations, we need to find sets of weights for each situation.

The desirability of trade-offs between criteria is context-dependent. In general, the properties that determine the quality of a solution are incommensurable. For example, the criteria may only be ordered (on an ordinal scale), with those criteria that determine the feasibility of a solution being greatly preferred to those properties that are merely desirable. It is difficult to see how to convert such problems into a multi-criterion optimization problem without making ad hoc assumptions. It is also far from clear that human behavior solely optimizes on a single criterion.

Rather than attempt to design a weighted-sum cost function, it is often more natural to formulate such problems in terms of a set of constraints that a solution should satisfy. We allow constraints to be prioritized, that is, it is more important to satisfy some constraints than others, and soft, that is, constraints are not absolute and can be satisfied to a greater or lesser degree. Such a framework is more general in admitting both optimization problems (e.g., minimization constraints) and satisficing problems (e.g., upper-bound constraints), which cannot be modeled by simply minimizing weighted-sum cost functions. Vicente

(1998) suggests ways in which such constraints can be analyzed as part of a work domain analysis.

This approach to working with constraints provides a way for more clearly specifying problem-solving tasks and more precisely evaluating the resulting solutions. There is a straightforward correspondence between the *real problem* and the constraints passed to the problem-solver. A solution can be characterized as satisfying some constraints (to a greater or lesser degree) and only partially satisfying or not satisfying others. By annotating solutions with the constraints they satisfy, the implications of adopting or executing the current best solution are immediately apparent. The annotations also facilitate the integration of the problem-solver into the architecture of an agent or a decision-support system (see for example, Logan & Sloman, 1998). If a satisfactory solution cannot be found, the degree to which the various constraints are satisfied or violated by the best solution found so far can be used to decide whether to change the order of the constraints, relax one or more constraints, or even redefine the goal, before making another attempt to solve the problem.

The ordering of constraints blurs the conventional distinction between absolute constraints and preference constraints. All constraints are preferences that the problem-solver will try to satisfy, trading off slack on a more important constraint to satisfy another, less important constraint.

The A* search algorithm is ill-suited to dealing with problems formulated in terms of constraints. Researchers at Birmingham have therefore developed a generalization of A* called A* with Bounded Costs (ABC; Alechina & Logan, 1998; Logan & Alechina, 1998), which searches for a solution that best satisfies a set of prioritized soft constraints.

The utility of this approach and the feasibility of the ABC algorithm have been illustrated by an implemented route planner that is capable of planning routes in complex terrain satisfying a variety of constraints. This work was originally motivated by difficulties in applying classical search techniques to agent-route planning problems. However, the problems identified with utility-based approaches, and the proposed solutions, are equally applicable to other search problems.

5.3 Psychologically Inspired Architectures

We review here several psychologically inspired cognitive architectures that were not covered by Pew and Mavor (1998). These architectures are interesting because (1) they are psychologically plausible, (2) some of them provide examples of how emotions and behavioral moderators can be included, and (3) several illustrate that better interfaces for creating cognitive models are possible.

5.3.1 Elementary Perceiver and Memoriser

The Elementary Perceiver And Memoriser (EPAM) is a well-known computer model of a wide and growing range of memory tasks. The basic ideas behind EPAM include mechanisms for encoding chunks of information into long-term memory by constructing a discrimination network. The EPAM model has been used to simulate a variety of psychological regularities, including the learning of verbal material (Feigenbaum & Simon, 1962, 1984) and expert digit-span memory (Richman, Staszewski, & Simon, 1995). EPAM has been expanded to use visuo-spatial information (Simon & Gilmartin, 1973).

EPAM organizes memory into a collection of chunks, where each chunk is a meaningful group of basic elements. For example, in chess, the basic elements are the pieces and their locations; the chunks are collections of pieces, such as a king-side pawn formation. These chunks are developed through the processes of discrimination and familiarization. Essentially, each node of the network holds a chunk of information about an object in the world. The nodes are interconnected by links into a network with each link representing the result of applying a test to the object. When trying to recognize an object, the tests are applied beginning from the root node, and the links are followed until no further test can be applied. When a node is reached, if the stored chunk matches that of the object then familiarization occurs. The chunk's resolution is then increased by adding more details of the features in that object. If the current object and the chunk at the node reached differ in some feature, then discrimination occurs, which adds a new node and a new link based on the mismatched feature. Therefore, with discrimination, new nodes are added to the discrimination network; with familiarization, the resolution of chunks at those nodes is increased.

The Chunk Hieraracy and REtrieval STructures (CHREST; de Groot & Gobet, 1996; Gobet & Simon, 1996b) is one of the most current theories of memory developed from the ideas in EPAM. Gobet and Simon (2000) present a detailed description of the present version of CHREST and report simulations on the role of presentation time in the recall of game and random chess positions. As in the earlier chunking theory of Chase and Simon (1973), CHREST assumes that chess experts develop a large EPAM-like net of chunks during their practice and study of the game. In addition, CHREST assumes that some chunks, which recur often during learning, develop into more complex retrieval structures (templates) with slots for variables that allow a rapid encoding of chunks or pieces.

EPAM and its implementations are important to consider because they fit a subset of regularities in memory very well. This at least serves as an example for other theories and architectures to emulate. It may also be possible to include the essentials of EPAM in another system, such as Soar or ACT-R, extending the scope of both approaches.

5.3.2 Neural Networks

Pew and Mavor (1998, chap. 3) review neural networks. Here, therefore, we only provide some further commentary, introduce some more advanced concepts, and note a few further applications.

Connectionist systems have demonstrated the ability to learn arbitrary mappings. Architectures such as the Multi-Layer Perceptron (MLP) are capable of being used as a black box that can learn to recognize a pattern of inputs as a particular situation. This requires supervised training and may involve heavy computational resources to arrive at a successful solution using the back-propagation algorithm. Training can be continued during performance as a background task, and thus, an entity could have an ability to learn during action based on this approach. Recognition performance is relatively rapid and a multi-layer perceptron might be used to model a reaction mechanism in which a combatant responds to coming under fire, or spotting the presence of the enemy, for example. It might also be used to activate particular aspects of military doctrine depending on the current circumstances.

Recurrent nets such as the Elman (1991) net have the ability to generate sequences of tokens as output. These seem to offer some promise of detecting an input situation and producing a series of behavioral actions as a response. This behavior of recurrent nets may be useful for modeling the reactive behavior of an entity over a short time period, while a symbolic cognitive model is used for the higher-level cognitive processes that occur over a longer time span.

5.3.3 Sparse Distributed Memories

Subtle issues such as the *tip-of-the-tongue* phenomena (Koriat & Lieblich, 1974) and the fact that we know if we know something (feeling of knowing) before becoming aware of the answer are not often modeled (although, see Schunn, Reder, Nhouyvanisvong, Richards, & Stroffolino, 1997, for a counter example). These effects may be captured using memory models such as Kanerva's (1988) Sparse Distributed Memory (SDM), which has been put forward as a plausible model of brain architecture, particularly the cerebellum, as well as by Albus's (1971) Cerebellar Model Arithmetic Computer (CMAC).

The way in which a combatant's experience of the world is stored and modeled is important. An SDM seems to offer powerful human-like ways of recalling nearest matches to present experience in a best-first manner. They have the interesting property of storing memories such that recall works by finding the best match to imperfect data. They are also a natural way of storing sequences. They exploit interesting mathematical properties of binary metric spaces with a large number of dimensions. It is intriguing that SDMs have the human-like properties that they "know if they know" something before the retrieval process is complete. They also exhibit the tip-of-the-tongue phenomenon and replicate the human ability to recall a sequence or tune given the first few items or notes. They can also learn actuator sequences that might be used in muscle control or reflex patterns of behavior. This can even be seen as a kind of thinking by analogy that has a uniquely human-like ability to find a close match rapidly without exhaustive or even significant time spent in search.

5.3.4 PSI and Architectures That Include Emotions

PSI is a relatively new cognitive architecture designed to integrate cognitive processes, emotions, and motivation (Bartl & Dörner, 1998). The architecture includes six motives (needs for energy, water, pain avoidance, affiliation, certainty, and competence). Cognition is modulated by these motive/emotional states and their processes. In general, PSI organizes its activities similar to Rasmussen's (1983) hierarchy: first, it tries highly automatic skills if possible, then it skips to knowledge-based behavior, and as its *ultima ratio* approach it uses trial-and-error procedures. It is one of the only cognitive architectures that we know about that takes modeling emotion and motivation as one of its core tasks. Its source code, in Delphi Pascal, is available (www.uni-bamberg.de/ppp/insttheopsy/psi-software.html).

A model in the PSI architecture has been tested against a set of data taken from a dynamic control task. The model performed the same task and its number of control actions was within the range of human behavior. Its predictions of summary scores were outside the range of human behavior—the model was less competent (Detje, 2000)—but single subjects can be modeled by varying starting parameters (Dörner, 2000). In such a complex task as the "Island" scenario some people will use meta-cognition to improve their performance (particularly if they are encouraged to think aloud as they were in Detje's study). The same data could reveal that these subjects profit from meta-cognition and that their behavior then differs from what is implemented currently in PSI (see Bartl, 2000, for a more detailed explanation).

This model needs to be improved before it matches human emotional data as well as other cognitive models match non-emotional data. It is, however, one of the few models of emotion compared with data.

The PSI architecture is currently incomplete, which raises interesting questions about how to judge a nascent architecture. PSI does not have a large enough user community and has not been developed long enough to have a body of regularities to be compared with let alone adjusted to fit. How can PSI be compared with the older architectures with existing tutorials, user manuals, libraries of models, and example applications?

Several other models of emotions and architectures that use emotions have been created. Reviews of emotional models (Hudlicka & Fellous, 1996; Picard, 1997) typically present models and architectures that have not been compared and validated against human data. There appears to be one other exception, an unpublished PhD thesis by Araujo at the University of Sussex (cited in Picard, 1997). Some of us are attempting to add several simple emotions to ACT-R (Belavkin, 2001; Belavkin et al., 1999) and validate the model by comparing the revised model with an existing model and comparable data (G. Jones, Ritter, & Wood, 2000).

5.3.5 COGENT

COGENT is a design environment for creating cognitive models and architectures (Cooper & Fox, 1998). It allows the user to draw box-and-arrow diagrams to structure and illustrate the high-level organization of the model and to fill in the details of each box using one or a series of dialogue sheets. The boxes include inputs, outputs, memory buffers, processing steps, and even production systems as components.

COGENT's strengths are that it is easy to teach, the displays provide useful summaries of the model that help with explanation and development, and the environment is fairly complete. It appears possible to reuse components on the level of boxes. COGENT's weaknesses are that it is fairly unconstrained; for large systems it may be unwieldy; and it might not interface well to external simulations.

COGENT also shows that cognitive modeling environments can at least appear more friendly. The results of its graphic interface routinely appear in talks as model summaries. The interface is also quite encouraging to users, allowing them to feel that they can start working immediately.

5.3.6 Hybrid Architectures

Hybrid architectures are architectures that typically include symbolic and non-symbolic elements. A more general definition would be architectures that include major components from multiple architectures.

Hybrid architectures are mentioned briefly by Pew and Mavor (1998, pp. 108-110). Work has continued in this area with some interesting results. LICAI (Kitajima & Polson, 1996; Kitajima, Soto, & Polson, 1998), for example, models how people explore and use interfaces based on a theory of how Kintsch's (1998) schemas receive activation. The U.S. Office of Naval Research (ONR) has sponsored a research program on hybrid architectures (Gigley & Chipman, 1999). This has given rise to some new, interesting hybrid architectures (e.g., Sun, Merrill, & Peterson, 1998; Wang, Johnson, & Zhang, 1998).

Perhaps the most promising hybrids are melding perception components across cognitive architectures. The EPIC (Kieras & Meyer, 1997) architecture's perception and action component has been merged with ACT-R's perceptual-motor component, ACT-R/PM (Byrne, 2001; Byrne & Anderson, 1998) and with Soar (Chong, 2001). This has led to direct reuse and unification. Similar results have been found with the Nottingham functional interaction architecture being used by Soar and ACT-R models (Bass et al., 1995; Baxter & Ritter, 1996; Ritter et al., 2000; G. Jones et al., 2000).

5.4 Knowledge-Based Systems and Agent Architectures

Agent architectures will be important within synthetic environments for modeling autonomous vehicles and for exploring the doctrine of autonomous vehicles. Most principled agent architectures have historical roots in distributed artificial intelligence. For several decades, distributed AI has been tackling essentially the same problem as Knowledge-Based Systems (KBS) research, namely, how to produce efficient problem-solving behavior in software. The main concept that brings agency and KBS together is the idea of operation at the knowledge level as described by Newell (1982).

The behavioral law used by an observer to understand the agent at the knowledge level is the principle of maximum rationality (Newell, 1982), which states, "If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action." The modeling of intelligent artificial systems at the knowledge level, that is, with no reference to details of implementation, is a key principle in KBS construction. It is also at the heart of many assumptions in the tradition of explaining human behavior.

Nwana (1996) claims that an important difference between agent-based applications and other distributed computing applications is that agent-based applications operate typically at the knowledge level, whereas distributed computing applications operate at the symbol level. At the symbol level, the entity is seen simply as a mechanism acting over symbols, and its behavior is described in these terms.

The theoretical links between the motivations behind KBS and agent research can be seen in the main approaches taken to the definition of software agency. Ascriptional agency attempts to create convincing human-like behaviors in software in the belief that this will produce programs that are easy to interact with. This work can be seen as paralleling the expert behavioral modeling approach that is currently widely espoused in the KBS community. The Belief-Desire-Intention (BDI) agents focus on the concept of intentionality—the mental attitudes of the agent. BDI models have been successfully implemented in systems such as the DESIRE framework (Brazier, Dunin-Keplicz, Treur, & Verbrugge, 1999) and the JAVA Agent Compiler and Kernel (JACK) component system (Busetta, Howden, Rönnquist, & Hodgson, 1999a; Busetta, Rönnquist, Hodgson, & Lucas, 1999b).

JACK is an extension to JAVA. It includes a JAVA library and a compiler that takes a JAVA program with embedded JACK statements. A JAVA compiler expands/incorporates the JACK statements to create a runnable JAVA program. These statements implement a BDI architecture, while allowing JAVA statements to extend and implement them. The statements include commands like @achieve(*condition*, *event*), which subgoals on *event* if *condition* is not found to be true.

The resulting program instantiates a BDI agent. Its BDI architecture is made up of beliefs represented with a database; desires represented as events that can trigger plans; and intentions represented through these plans. For example, a fact may come in from perception and match a desire, that of putting new facts into the database. This may result in further desires being matched and intentions (plans) leading to behaviors. Further information is available at the JACK developer's website (www.agent-software.com.au).

Reviews of the agent literature (Etzioni & Weld, 1995; Franklin & Graesser, 1997; Wooldridge & Jennings, 1995)[2] reveal that, when attempting to define agency as dependent on the possession of a set of cardinal attributes, many of the attributes suggested could also be seen as characteristic of behavior that is best explained at the knowledge level. These include abstraction and delegation, flexibility and opportunism, task orientation, adaptivity, reactivity, autonomy, goal-directedness, flexibility, collaborative and self-starting behavior, temporal continuity, knowledge-level communication ability, social ability, and cooperation.

Both agent systems and KBSs are moving in the direction of modular components of expertise as a response to the problems of knowledge use and reuse to promote intelligent behavior in software. Domain ontologies form a significant subset of these KBS components. Increasingly, multi-agent systems are being produced that use such domain ontologies to facilitate agent communication at the knowledge level, for example, the agent network created as part of the Infosleuth architecture (Jacobs & Shea, 1996). Some agent systems also draw explicitly on models of problem-solving expert behavior developed in KBS research. The internet-based Multi-agent Problem Solving (IMPS) architecture (Crow & Shadbolt, 1998) uses software agency as a medium for applying model-driven knowledge engineering techniques to the internet. It involves software agents that can conduct structured online knowledge acquisition using distributed knowledge sources. Agent-generated domain ontologies are used to guide a flexible system of autonomous agents driven by problem-solving models.

[2] For online information about examples and related U.S. programs, see www.darpa.mil/ito/ResearchAreas.html and www.nosc.mil/robots/air/amgsss/mssmp.html.

5.5 Architectural Ideas Behind the Sim_Agent Toolkit[3]

Since the early 1970s, Sloman and his colleagues have been attempting to develop requirements and designs for an architecture capable of explaining a wide variety of facts about human beings and other intelligent agents. Sloman's ideas about cognitive architectures and the theoretically based agent architecture toolkit (Sim_Agent) provide useful lessons about architectural toolkits and about process models of emotions. Further information is available at the CogAff website (www.cs.bham.ac.uk/~axs/cogaff.html).

5.5.1 Cognition and Affect

A human-like information processing architecture includes many components performing different functions all of which operate in parallel, asynchronously. This is not the kind of low-level parallelism found in neural nets (although such neural mechanisms are part of the infrastructure). Rather there seem to be many functionally distinct modules performing different sorts of tasks concurrently, a significant proportion of them are concerned with the monitoring and control of bodily mechanisms, for example, posture, saccades, grasping, temperature control, daily rhythms, and so on.

The very oldest mechanisms in the human architecture are probably all reactive in the sense described in various recent papers (e.g., Sloman, 2000). The key feature of reactivity is the lack of "what-if" reasoning capabilities, with all that entails, including the lack of temporary workspaces for representations of hypothesized futures (or past episodes); the lack of mechanisms for stored factual knowledge (generalizations and facts about individuals) to support the generation of possible futures, possible actions, and likely consequences of possible actions; and the lack of mechanisms for manipulating explicit representations.

Both reactive and deliberative mechanisms require perceptual input and can generate motor signals. However, to function effectively, both perceptual and action subsystems may have evolved new layers of abstraction to support the newer deliberative processes, for example, by categorizing both observed objects and events at a higher level of abstraction, and allowing higher-level action instructions to generate behavior in a hierarchically organized manner. More generally, different subsystems use information for different purposes so that a number of different processes of analysis and interpretation of sensory input occur in parallel, extracting different affordances from raw data from the optic array. Recent work by brain scientists on ventral and dorsal visual pathways are but one manifestation of this phenomenon.

The interactions between reactive and deliberative layers are complex and subtle, especially as neither is in charge of the other, though at times either can dominate. Moreover, the division is not absolute: information in the deliberative system can sometimes be transferred to the reactive system (e.g., via drill and practice learning), and information in the reactive system can sometimes be decompiled and made available to deliberative mechanisms (though this is often highly error-prone).

[3] This section was drafted by Aaron Sloman and revised by the authors.

For reasons explained in various papers available in the CogAff FTP site, it is possible to conjecture that at a much later evolutionary stage a third class of mechanism developed, again using and redeploying mechanisms that had existed previously. The new type of mechanism, which has been provisionally labeled "meta-management," provides the ability to do for internal processes what the previous mechanisms did for external processes: namely it supports monitoring, evaluation, and control of other internal processes, including, for instance, thinking about how to plan, or planning better ways of thinking. For example, a deliberative system partly driven by an independent reactive system and sensory mechanisms can unexpectedly acquire inconsistent goals. A system with meta-management can notice and categorize such a situation, evaluate it, and perhaps through deliberation or observation over an extended period, develop a strategy for dealing with such conflicts.

Similarly, meta-management can be used to detect features of thinking strategies and, perhaps in some cases, notice flaws or opportunities for improvement. Such a mechanism (especially in conjunction with an external language) also provides a route for absorption of new internal processes from a culture, thereby allowing transmission between generations of newly acquired information without having to wait for new genetic encodings of that information to evolve. Through internal monitoring of sensory buffers, the extra layer adds a kind of self-awareness that has been the focus of discussions of consciousness, subjective experience, qualia, etc. As with external processes, the monitoring, evaluation, and re-direction of internal processes is neither perfect nor total and, as a result, mistakes can be made about what is going on, inappropriate evaluations of internal states can occur, and attempts to control processing may fail, for example, when there are lapses of attention despite firm intentions.

Another feature of meta-management is its ability to be driven by different collections of beliefs, attitudes, strategies, and preferences, in different contexts, explaining how a personality may look different at home, driving a car, in the office, etc. Besides the three main concurrent processing layers (reactive, deliberative, and meta-management) identified above that others have found evidence for, a number of additional specialized mechanisms are needed, including: mechanisms for managing short- and long-term goals, a variety of long- and short-term memory stores, and one or more global alarm systems capable of detecting a need for rapid global re-organization of activity (freezing, fleeing, attacking, becoming highly attentive, etc.), and also producing that re-organization.

For instance, whereas many people have distinguished primary and secondary emotions (e.g., Damasio, 1994), Sloman and his colleagues have proposed a third type, tertiary emotions, sometimes referred to as *perturbances* (Sloman, 1998a; Sloman & Logan, 1999). Primary emotions rely only on the reactive levels in the architecture. Secondary emotions require deliberative mechanisms. Tertiary emotions are grounded in the activities of meta-management, including unsuccessful meta-management. There are other affective states concerned with global control, such as moods, which also have different relationships to the different layers of processing. Many specific states that are often discussed but very unclearly defined, such as arousal, can be given much clearer definitions within the framework of an architecture that supports them.

It looks as if various subsets of the capabilities described here arising out of the three layers and their interactions can be modeled in the architectures developed so far, for example, Soar, ACT-R/PM, Moffatt and Frijda's Will architecture (2000), and the various logic-based models that dominate the ATAL (Architectures, Theories and Languages) series of workshops (e.g., Wooldridge, Mueller, & Tambe, 1996, also see mas.cs.umass.edu/atal/), and books like Wooldridge and Rao (1999).

However, only small subsets of these capabilities can be modeled at present. Any realistic model of human processing needs to be able to cope with contexts including rich bombardment with multi-modal sensory and linguistic information; where complex goals and standards of evaluation are constantly interacting; where things often happen too fast for fully rational deliberation to be used; where everything that occurs does not always fall into a previously learned category for which a standard appropriate response is already known; where decisions have to be taken on the basis of incomplete or uncertain information; and where the activity of solving one problem or carrying out one intricate task can be subverted by the arrival of new factual information, new orders, or new goals generated internally as a side-effect of other processes.

Where the individual is also driving a fast-moving vehicle or is under fire then it is very likely that a huge amount of the processing going on will involve the older reactive mechanisms, including many concerned with bodily control and visual attention. It may be some time before we fully understand the implications of such total physical immersion in stressful situations, including the effects on deliberative and meta-management processes. (For example, fixing attention on a hard planning problem can be difficult if bombs are exploding all around you. Can our models explain why?)

5.5.2 Sim_Agent and CogAff

Sloman and his colleagues' general architectural toolkit, the Sim_Agent Toolkit, allows them to explore a variety of new ideas about complex architectures. It is not an architecture, but a steadily developing toolkit for exploring architectures.

The CogAff architecture provides a schema, based on a 3 by 3 grid that provides a framework for describing specific architectures according to the grid components present, their control relationships, and how information flows between them. H-CogAff is a specific human-like version that is a particularly rich special case. Other special cases include various kinds of purely reactive (i.e., non-deliberative) architectures (perhaps insects or reptiles), Brooks' subsumption architectures, purely deliberative architectures (lots of old AI systems, early versions of Soar and ACT), and so on.

Sloman and his colleagues also wanted a toolkit that supported exploration of a number of interacting agents (and physical objects, etc.) where each agent has a variety of very different mechanisms running concurrently and asynchronously, yet influencing one another. They also wanted to be able to very easily change the architecture within an agent, change the degree and kind of interaction between components of an agent, and speed up or slow down the processing of one or more sub-mechanisms relative to others (Sloman, 1998b). In particular, they wanted to be able to easily combine different types of symbolic mechanisms and also sub-symbolic mechanisms within one agent. The toolkit was also required to support rapid prototyping and interactive development with close connections between internal processes and graphic displays.

Because other toolkits did not appear to have the required flexibility and tended to be committed to a particular type of architecture, Sloman and his colleagues built their own toolkit, which has been used for some time at the University of Birmingham and DERA, Malvern. Their toolkit is described briefly in Sloman and Logan (1999) and in more detail in the online documentation at the Birmingham Poplog FTP site (ftp.cs.bham.ac.uk/pub/dist/poplog/). The code and documentation are freely available online. The toolkit runs in Pop-11 in the Poplog system (inherently a multi-language AI system, so that code in Prolog, Lisp, or ML can also be included in the same process). Poplog has become freely available (www.cs.bham.ac.uk/research/poplog/freepoplog.html).

At present, Sloman does not propose a specific overarching architecture as a rival to systems like Soar or ACT-R. He feels that not enough is yet known about how human minds work and, consequently, any theory proposing *the* architecture is premature. Instead, he and his group have been exploring and continually refining a collection of ideas about possibly relevant architectures and mechanisms. Although the ideas have been steadily developing, Sloman and his colleagues do not believe that they are near the end of this process. So, although one could use a label like *CogAff* to refer to the general sort of architecture they are currently talking about, it is not a label for a fixed design. Rather CogAff should be taken to refer to a high-level overview of a class of architectures in which many details still remain unclear. The CogAff ideas are likely to change in dramatic ways as more is learned about how brains work, about ways in which they can go wrong (e.g., as a result of disease, aging, brain damage, addictions, stress, abuse in childhood, etc.), and how brains differ from one species to another, or one person to another, or even within one person over a lifetime.

The toolkit is still being enhanced. In the short term, they expect to make it easier to explore architectures including meta-management. Later work will include better support for sub-symbolic spreading activation mechanisms and the development of more reusable libraries, preferably in a language-independent form.

5.5.3 Summary

The Sim_Agent toolkit and the goals its developers have for it have some commonalties with other approaches. The need for a library of components is acknowledged. They emphasize that reactive behaviors are necessary and desirable, and that the emotional aspects arise out of the reactive mechanisms. It provides a broad range of support for testing and creating architectures. The toolkit provides support for reflection as a type of meta-learning. Other architectures will need to support reflection as well, particularly where the world is too fast-paced for learning to occur during the task (John, Vera, & Newell, 1994; Nielsen & Kirsner, 1994).

The features that the toolkit supports help define a description of architectural types. The capabilities that can be provided, from perception to action and from knowledge to emotion, provide a way of describing architectures.

The major drawback is that none of the models or libraries created in Sim_Agent have been compared with human data directly. In defense of this lack of comparison, Sloman claims that the more complex and realistic an architecture becomes, the less sense it makes to test it directly. Instead, he claims that the architecture has to be tested by the depth and variety of the phenomena it can explain, like advanced theories in physics, which also cannot be tested directly.

5.6 Engineering-Based Architectures and Models

There is a history of studying process control in and near industrial engineering that includes studying human operators. This approach is not (yet) part of mainstream psychology, and Pew and Mavor (1998) do not make many references to work in this field.

If tank operators and ship captains can be viewed as running a process, and we believe they can, there is a wide range of behavioral regularities referenced and modeled in engineering psychology that can be generalized and applied to other domains. Major contributions in this area include Reason's (1990) book on errors, Rasmussen's skill hierarchy (1983), the Cognitive Reliability and Error Analysis Method (CREAM) methodology for analyzing human performance (Hollnagel, 1998), and numerous studies characterizing the strengths and weaknesses of human operator behavior (de Keyser & Woods, 1990; Sanderson, McNeese, & Zaff, 1994).

Engineers have also created intelligent architectures. These architectures have almost exclusively been used to create models of users of complex machinery, ranging from nuclear power plants to airplanes. The models are often, but not always, tied to simulations of those domains. Their approach is generally more practical. They are more interested in approximate timing and the overt behavior than in detailed mechanisms. These developers appear to be less interested in the internal mechanisms giving rise to behavior as long as the model is usable and approximately correct.

These models of operators include models of nuclear power plant operators, the Cognitive Simulation Model (COSIMO; Cacciabue et al., 1992), and the Cognitive Environment Simulation (CES; Woods, Roth, & Pople, 1987). AIDE (Amalberti & Deblon, 1992) is a model of fighter pilot behavior; the Step Ladder Model or Skill-based, Rule-based, Knowledge-based model is a generally applicable framework, originally formulated in electronics troubleshooting (e.g., Rasmussen, 1983).

We will also look at a few operator models in more detail.

5.6.1 APEX[4]

APEX (Freed & Remington, 2000; Freed et al., 1998; John et al., 2002) is a set of tools for simulating human performance when interacting with interfaces to perform tasks similar to MIDAS (Laughery & Corker, 1997). The main driver for APEX is the need to model behavior in environments, such as air traffic control and commercial jet flight decks, and to help engineers design usable systems in these domains

Powerful action-selection mechanisms of the sort developed by artificial intelligence researchers are used to cope adaptively with time-pressure and uncertainty, and to coordinate the execution of multiple tasks (i.e., strategic multi-tasking). Usability is taken very seriously (Freed & Remington, 2000). A high-level modeling language is included. Applications of APEX have included time-analysis of skilled behavior, partially-automated

[4] Comments from Michael Freed were helpful in preparing this section.

human-factors design analysis, and creation of artificial human participants in large-scale simulations.

This general approach has proven successful in allowing APEX to automate the CPM-GOMS HCI analysis method (John & Kieras, 1996) and for reconstructing incidents involving human error in a way that promise eventual error-prediction capabilities. As much as it implements CPM-GOMS, it inherits CPM-GOMS' empirical support. Consistent with the needs of domains in which APEX has been most frequently used, the action-selection architecture emphasizes capabilities having to do with multi-task management, especially regarding concurrency control and strategic task management.

APEX was created by Freed as part of his doctoral dissertation and continues to be developed by researchers at NASA Ames Research Center and elsewhere. They are explicitly concerned about building a community of users to share ideas and models. Further information, including APEX itself, is available through search engines.

APEX is probably best described as an engineering model because it has been designed to serve engineering goals. APEX is interesting because it models the whole operator, from perception to action, and the model often interacts with fairly complete and complex simulations, and can make very detailed predictions easily. It does not yet include learning, and the complex results past CPM-GOMS could be tested more, but the full toolset suggests that interface design tools based on cognitive models are now possible.

5.6.2 Simplified Model of Cognition and Contextual Control Model

The Simplified Model of Cognition (SMoC) (Hollnagel & Cacciabue, 1991) is an extension of Neisser's (1976) perceptual cycle and describes cognition in terms of four essential elements: (1) observation/identification, (2) interpretation, (3) planning/selection, and (4) action/execution. Although these are normally linked in a serial path, other links are possible between the various elements. The small number of cognitive functions in SMoC reflects the general consensus of opinion that has developed since the 1950s on the characteristics of human cognition. The fundamental features of SMoC are the distinction between observation and inference (overt vs. covert behavior), and the cyclical nature of cognition (cf. Neisser, 1976).

SMoC was formulated as part of the System Response Generator (SRG) project (Hollnagel & Cacciabue, 1991). SRG was a software tool developed to study the effect of human cognition (specifically actions and decision making) on the evolution of incidents in complex systems.

The Contextual Control Model (CoCoM; Hollnagel, 1993) is an extension of the SMoC, and addresses the issues of modeling both competence and control. In most models the issue of competence is supported by a set of procedures or routines that can be employed to perform a particular task when a particular set of pre-defined conditions obtains. CoCoM further proposes that there are four overlapping modes of control—influenced by knowledge and skill levels—that also influence behavior:

- Scrambled control: where the selection of the next action is unpredictable. This is the lowest level of control.
- Opportunistic control: where the selection of the next action is based on the current context without reference to the current goal of the task being performed.
- Tactical control: where performance is based on some form of planning.
- Strategic control: where performance takes full account of higher-level goals. This is the highest level of control.

The transition between control modes depends on a number of factors, particularly the amount of subjectively available time and the outcome of the previous action. These two factors are interdependent, however, and also depend on aspects such as the task complexity and the current control mode.

CoCoM has been used in the development of the CREAM (Hollnagel, 1998) within the field of human reliability analysis. The CREAM is a method for analyzing human performance when working with complex systems. It can be used in both the retrospective analysis of accidents and events, and in predicting performance for human reliability assessment. Extending the CREAM is presented below as a useful project.

5.6.3 Summary

These engineering-based architectures suggest that engineering models can provide useful behavior even when the internal mechanisms are not fully tested or perhaps even plausible. These architectures suggest that some of the difficulty in creating the architectures is due to the implicit and explicit knowledge that psychologists bring with them regarding plausibility. We believe psychologists' domain knowledge leads to more accurate models but slower development.

5.7 Summary of Recent Developments for Modeling Behavior

This chapter has reviewed several architectures. These architectures and their applications show that it is becoming increasingly possible to create plausible and useful architectures based on a variety of approaches.

An agreed, formal scheme for classifying architectures would be useful. This ideal system classification would note the sorts of tasks that each architecture performs best, supporting users to choose an architecture for a particular task. The best that we have found is Table 3.1 in Pew and Mavor (1998, pp. 98-105). Our Table 5.1 provides a summary of the architectures presented here in that same format as a supplement to their table. We have included all relevant information of which we are aware for each architecture. In most cases the developers of the architectures have helped complete their entry in this table. Another approach for classifying architectures is available from Logan (1998).

Developments in AI continue to be useful. The general AI methods discussed are not included in this table because they are not broad enough to be considered a cognitive architecture, but they are likely to be useful additions to architectures, either directly or indirectly. For example, genetic algorithms have been included in a proposed architecture

(Holland, Holyoak, Nisbett, & Thagard, 1986), and planning algorithms have been included as adjuncts to Soar (Gratch, 1998). These developments will help extend architectures by providing algorithms for inclusion within architectures, particularly hybrid architectures.

There are several interesting trends to note. One is that the diversity of architectures is not decreasing. New, fundamental ideas on which to base architectures has widened from simply problem solving. For example, EPAM is based on pattern recognition, and PSI and architectures created in the Sim_Agent Toolkit are based on ideas about emotions.

Another interesting trend is that some aspects of the architectures are starting to merge and be reused. The interaction aspects of EPIC have been reused by Soar and by ACT-R. The Nottingham Interaction Architecture is similar in some ways and getting similar reuse (e.g., Jones et al., 2000). These strands are becoming quite similar to each other (Byrne, Chong, Freed, Ritter, & Gray, 1999) and are quite likely to merge in the future.

The importance of model usability is becoming more recognized. COGENT provides an example of how easy a modeling tool should be to pick up and use. Similar developments with Soar and ACT-R are starting to emphasize reusable code, better documentation, and better tutorial materials. Other architectures will have to follow suit to attract users and to train and support their existing users. Newell (1990) wrote about the entry level (the bar) being raised as architectures develop through competition. It is interesting that usability is perhaps the first clear comparison level.

Table 5.1: Comparison of Architectures

			Submodels
	Architecture	Original purpose	Sensing and perception
1	EPAM	Model high-level perception, learning, and memory	Visual, auditory perceptual discrimination in real-time (assuming feature-based description of objects)
2	SDM	Simulation of cerebellum as a content-addressable memory	Can be used to recall the nearest stored memory to any encoded perceptual input
3	PSI	Explores interaction of cognition, motivation, and emotion to build an integrated model of human action regulation	Optical perception by “Hypercept” process that scans (simulated) environment for basic features. Raises hypotheses about sensory schemas to which features may belong and tests these hypotheses by subsequent scanning of environment (comparable to saccadic eye-movements). If pattern not recognizable, new schema generated
4	COGENT	Design environment for modeling cognitive processes	Input buffers that can be modified to represent vision and hearing
5	JACK as example of BDI architectures	Constitute an industrial-strength framework for agent applications	JAVA methods + inter-agent messaging
6	Sim_Agent Toolkit	Explores architectures using rapid prototyping	Defined by methods for each agent class.
7	Engineering-based models (e.g., APEX)	Provide models of humans in control loops	Varies, but exists for most models

Table 5.1: Comparison of Architectures (continued)

	Submodels		
	Working/ Short-Term memory	Long-Term Memory	Motor Outputs
1	4-7 slot STM; in some versions (e.g., EPAM-IV), more detailed implementation of auditory (Baddeley-like) STM & visual STM	Discrimination net. In recent versions, nodes of discrimination net used to create semantic net and productions	Eye movements, simple drawing behavior
2	Not modeled	Sparse Distributed Memory models related to PDP and neural-net memory models	Motor sequences can be learned. Nearest match memories can be sequences that could be behaviors
3	The *head* of a protocol memory that permanently makes a log of actions and perceptions	The remnants of logs decay with time. Strings of logs associated with need satisfaction or with pain will be reinforced and have a greater chance to survive and form a part of long-term memory than neutral sequences of events	Basic motor patterns (actions) combined to form complex sensory-motor-programs by learning (i.e., by reinforcement of the successful sensory-motor-patterns in logs)
4	Various types supported	Various types supported	Simple buffer representation of commands
5	Object-oriented structures (JAVA), plus relational modeling support (JACK)	All JAVA support including database interfaces etc. Support for data modeling in JAVA and C++ using JACOB (JACK Object Builder)	JAVA methods
6	List structures	List structures, rules, and arbitrary Pop-11 data structures. Can also use neural nets, if required	Defined by methods for each agent class
7	Usually simple, but extant	Usually simple, but extant	Usually extant, but usually not complex

Table 5.1: Comparison of Architectures (continued)

	Knowledge Representation	
Architecture	Declarative	Procedural
1 EPAM	Chunks, schemas (templates); using nodes in discrimination net	Productions using nodes in discrimination net
2 SDM	A sparse set of memory addresses where data *are* addresses	Memories naturally form sequences that could be considered procedures
3 PSI	Sensory and sensory-motor patterns consisting of pointer structures forming schemas. A schema includes information about more basal elements and relations of elements in space and time, including language patterns pointing to sensory and sensory-motor patterns (implementation in progress)	Sensory-motor-patterns forming automatisms
4 COGENT	Numbers, strings, lists, tuples, connectionist networks	Production rules, connectionist networks, Prolog
5 JACK as an example of BDI architectures	Object-oriented structures (JAVA), plus relational modeling support	JACK plans and JAVA methods
6 Sim_Agent Toolkit	List structures and arbitrary Pop-11 data structures (e.g., could be constrained to express logical assertions but need not be). Could use neural nets or other mechanisms	Rule sets and arbitrary Pop-11 procedures that can also invoke Prolog or external functions
7 Engineering-based models (e.g., APEX)	Varies, but usually simple	Varies, but usually simple. Many use some form of schemas

Table 5.1: Comparison of Architectures Covered (continued)

	Higher-Level Cognitive Functions			
	Learning	Planning	Decision Making	Situation Assessment
1	Chunking, creation of schemas, and production learning online (incremental) and stable against erroneous data	Connections between templates used in planning	Knowledge based	Overt and inferred
2	By incrementing weights across a probability distribution	Does not plan, but can remember plans	Iterative memory recall process	Can learn a set of assessments and generalize these
3	Associative and perceptual learning; operant conditioning: sensory-motor learning, learning goals (situations that allow need satisfaction) and aversions (situations or objects that cause needs)	Built-in hill-climbing procedure: action schemata (i.e., sensory-motor-patterns) are recombined to form new plans. If planning unsuccessful or impossible due to lack of information, trial-and-error procedures used to collect environmental information	Expectancy-value-principle	Built in as part of problem solving
4	Common methods within connectionist modules	Could be implemented in rule modules	Specific to module type. Can vary	None built in (users can specify)
5	None built in (users can specify as required by their architecture)	None built in (users can specify as required by their architecture)	Includes BDI computation model	Includes BDI computation model
6	None built in (e.g., Wright et al. 1996, included simple forms of deliberative mechanisms and meta-management)	None built in (users can specify as required by their architecture). Logan's A* with bounded constraints available, among others	None built in (users can specify as required by their architecture)	None built in (users can specify as required by their architecture)
7	Usually not extant	Varies, some models do well	Usually good; decision making domain of these models	Varies, often implicit

Table 5.1: Comparison of Architectures (continued)

	Multitasking	
Architecture	Serial/Parallel	Resource Representation
1 EPAM	Serial processing; learning done in parallel	Limited STM capacity, limited perceptual and motor resources (uses time parameters)
2 SDM	Fully parallel recall process, serial recall of sequences	Architecture too low-level for representation to be explicit
3 PSI	System tries to fulfill different needs (i.e., water, energy, pain-avoidance, etc.); interrupts goal-directed behavior to profit from unexpected opportunities	Allocation of time to run intention according to strength of underlying need and according to expectancy of success
4 COGENT	Modules can work in parallel, but information passed between them serially	Would vary with the knowledge included in modules
5 JACK as an example of BDI architectures	Supports multiple computational threads handled safely within the JACK Kernel—achieving atomic reasoning steps	Agents have time perception. Time can be real or simulated (dilated, externally synchronized, etc.)
6 Sim_Agent Toolkit	Discrete event simulation technique, with rule sets within each agent time-sliced, as well as different agents being time-sliced	Allocation of cycles per time-slice can be made for each rule set, or for each agent. No built-in memory resource limits. Will differ for each architecture type created
7 Engineering-based models (e.g., APEX)	Varies, sometimes explicit models	Varies. Those that interact with simulations more advanced

Table 5.1: Comparison of Architectures (continued)

	Goal/Task Management	Multiple Human Modeling	Implementation Platform
1	Bottom up + 1 main goal per task simulated	Potential through multiple EPAM modules	Mac, PC (any system supporting Common Lisp). Graphic environment supported only for Macintosh
2	None	None	UNIX (easily ported)
3	There is a steady competition of different needs/motives to rule. Strongest will win and inhibit others	Potential through multiple PSI models with different "personalities" by varying starting parameters. Multiple agents can run in same environment, see each other, interact, and, to a certain degree, communicate	Windows 95, 98, 2000, NT
4	None built in. Users can specify through module selection and programming	None	UNIX (X windows). Microsoft Windows
5	Built in. JACK Language includes: wait_for (condition), maintenance conditions, meta-level reasoning, etc.	Allows multiple agents, running together or distributed, to interact and communicate as a team or as adversaries. Extensions to the basic model (e.g., team models also allowed)	Runs on all platforms that support JAVA 1.1.3 or later. Graphic components (i.e., development environment) require JAVA 2 v 1.2 or later
6	None built in (users can s pecify as required by the architecture)	Toolkit allows multiple agents to sense one another, act on one another, and communicate with one another	Runs on any system supporting Poplog (and for graphics, X window system). Tested on Sun/Solaris, PC/Linux, DEC Alpha/UNIX. Should also run on other UNIXes and VAX VMS. Should work without graphics on Windows NT Poplog
7	Varies. Some advanced	Some have none; some work in teams	Varies. Not usually designed for dissemination

Table 5.1 Comparison of Architectures (continued)

Architecture	Implementation Language	Support Environment
1 EPAM	Common Lisp	Lisp programming + editing tools. Some graphic utilities for displaying eye movements, structure of discrimination tree, and task. Customized code used for each task modeled
2 SDM	C, JAVA	None
3 PSI	Pascal (Delphi 4)	Delphi 4 features
4 COGENT	Prolog	Graphic and textual editors
5 JACK as an example of BDI architectures	JAVA. JACK written in and compiles into pure JAVA	JACK Make utilities, and all available JAVA tools. JACK development environment (JDE) provides GUI for creating and editing agent structures. Further debugging and visualization tools under development
6 Sim_Agent Toolkit	Pop-11 (but allows invocation of other Poplog languages (Prolog, Common Lisp, Standard ML, & external functions, e.g., C)	Poplog environment, including VED/XVED, libraries, incremental compiler, etc
7 Engineering-based models (e.g., APEX)	Varies	Often simple

Table 5.1: Comparison of Architectures (continued)

	Validation	Comments
1	Extensive at many levels	EPAM models focus on single, specific information processing task at a time. Not yet scaled up to multitasking situations. Used in high-knowledge domains (e.g., chess, with about 300,000 chunks)
2	None	SDM should be seen as system component (e.g., good way of representing long-term memory for patterns and motor behaviors in larger system)
3	Achievement data and parameters of behavior compared between subjects and models in two different scenarios (BioLab and Island). Different human subjects can be modeled by varying parameters	
4	Would be by architecture. Some have been done by modeling previously validated models	
5	Would be by architecture. None known	
6	Would be by architecture. None known	
7	By model. Usually validated with expert opinion. Some may be compared with data	Wide range of models here